



# **Kompresja H.264 - wprowadzenie do standardu i problemy implementacyjne**

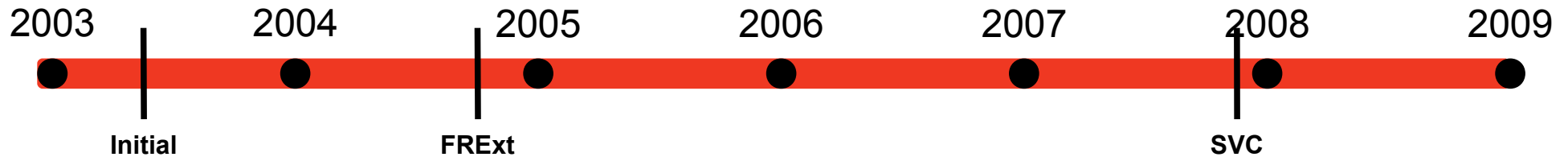
Adam Dawidziuk

[adam@cubiware.com](mailto:adam@cubiware.com)

**16 października 2008r.**

# Wprowadzenie

**ITU-T H.264 = MPEG-4 AVC = ISO/IEC 14496-10 = MPEG-4 Part 10**



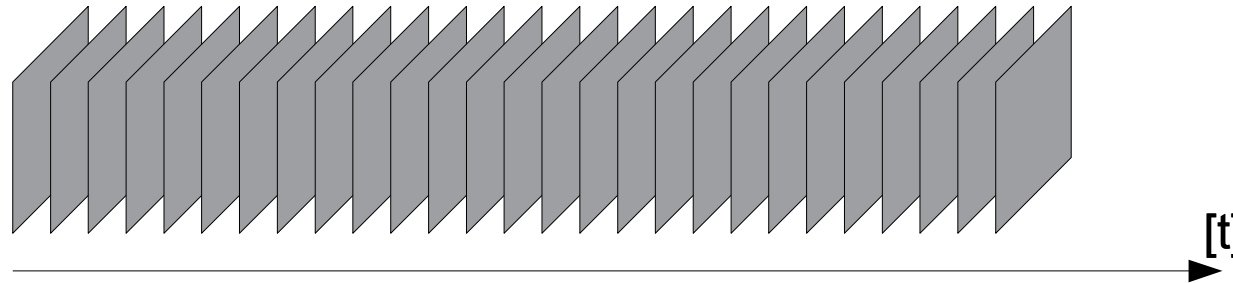
- **Zastosowania**

- "strumieniowanie" przez Internet (np. YouTube, kamierki internetowe, etc.)
- wideo-konferencje, wideo rozmowy
- nadzór wideo
- telewizja cyfrowa (DVB-C/T/H, IPTV)
- kino domowe (Blue-Ray/HD-DVD)
- kamery video, aparaty cyfrowe (nagrywanie filmów)
- archiwizacja (DV->H.264, x264, Nero, etc.)

- "Konkurencja": MPEG-2, H.263, MPEG-4 Part 2, VC-1, WMV9

# Słowniczek : na początek

- [“Niezakodowany”] strumień wideo: ciąg obrazów (bitmap 2D), zwykle **skorelowanych czasowo i przestrzennie**

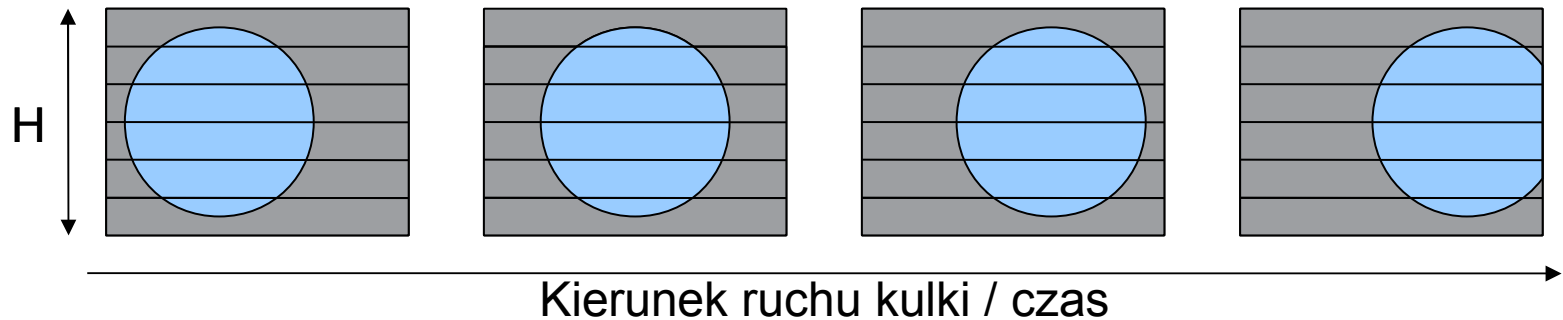


- Modelowanie – zmniejszanie redundancji przestrzennej i czasowej w obrazach
- Kodowanie – reprezentacja symboli/obiektów nowego źródła (o zmniejszonej redundancji) za pomocą kodu binarnego
- “Zakodowany” strumień bitowy: bitowa (“1D”) reprezentacja strumienia wideo po fazie modelowania i kodowania

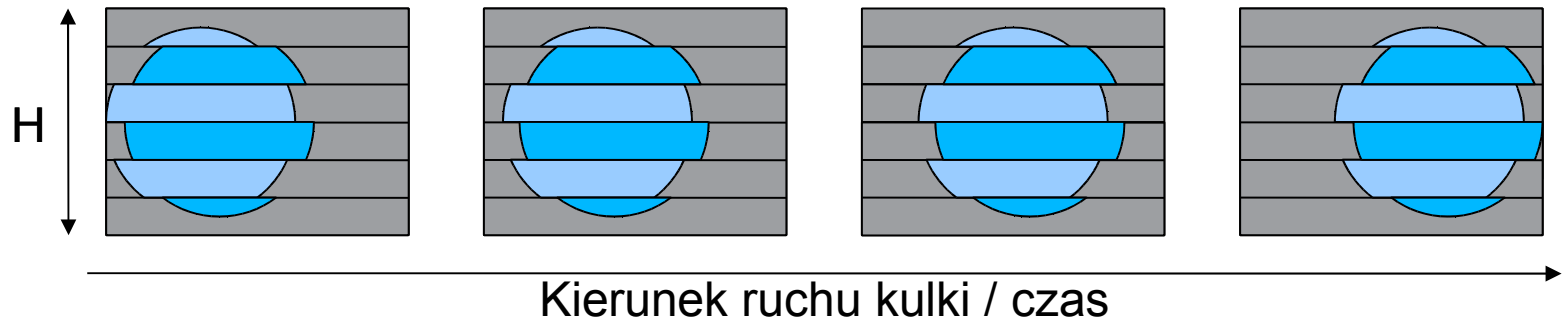
10010101110001001001011101010010101010010101.... [t]

# Słowniczek : ramki wideo

- Ramka wideo bez przeplotu – bitmapa (np. o rozmiarach  $W$  [pikseli] x  $H$  [linii]), której wszystkie piksele we wszystkich liniach pochodzą z “jednej” chwili czasowej.



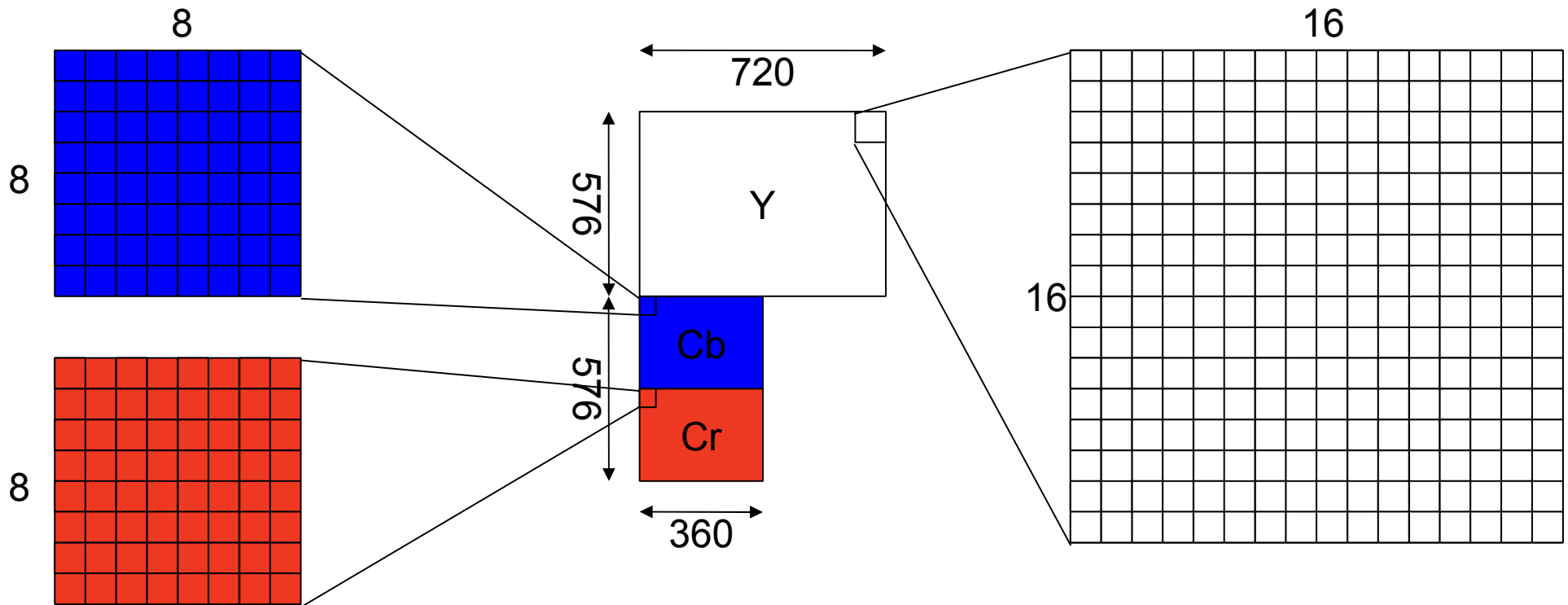
- Ramka wideo z przeplotem – bitmapa (np. o rozmiarach  $W$  [pikseli] X  $H$  [linii]), której piksele w liniach parzystych i nieparzystych pochodzą z dwóch różnych chwil czasowych.



- Ramki video zwykle są zakodowane w przestrzeni kolorów YCbCr 4:2:0 lub YCbCr 4:2:2

# Słowniczek : ramki wideo (cd)

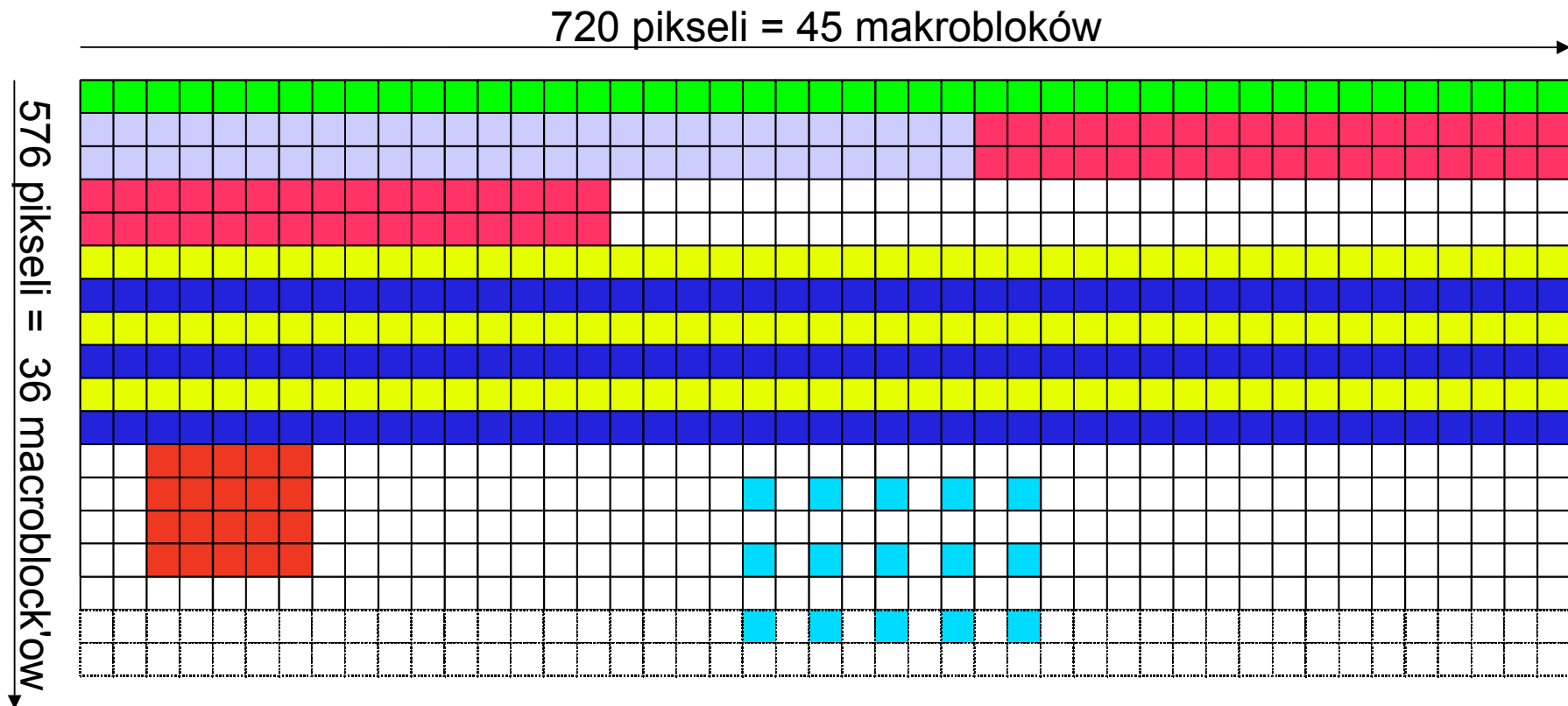
- Przykładowa reprezentacja ramki wideo w pamięci (PAL, 4:2:0):



- Makroblok (Y/luminancji) – obszar składowej Y obrazu o rozmiarach 16x16 pikseli
- Makroblok Cb/Cr – obszar składowych Cb/Cr obrazu o rozmiarach 8x8 pikseli (TODO: jak w HP422?)
- Kodowana ramka wideo musi mieć rozmiary składowej Y (w pionie i poziomie), które są wielokrotnością 16 (musi zawierać całkowitą liczbę makrobloków)

# Słowniczek : slice

- Slice – wycinek obrazu grupujący całkowitą wielokrotność makrobloków

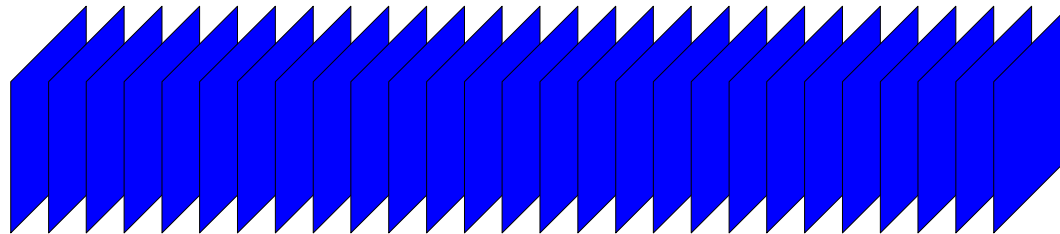


- Ograniczenia na ilość, rozmiar, położenie i konfigurację wzajemną slice'ów w zależności od profilu
- Każdy slice jest kodowany osobno (niekoniecznie w oderwaniu od reszty obrazu)

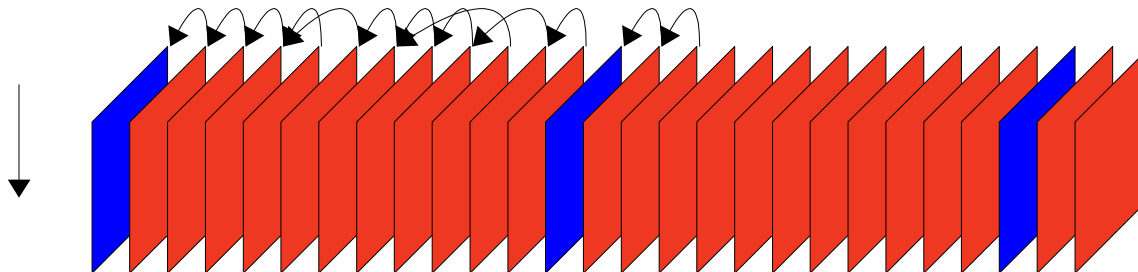
# Słowniczek – rodzaje ramek

- Ramki / slice'y mogą być I, P, B (tj. zawierać makrobloki typu I, P, B)

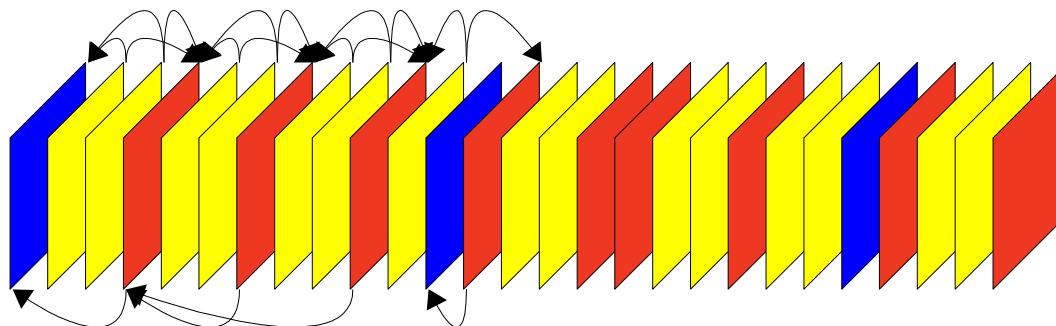
I - Intra coded – zdekodowanie makrobloku nie wymaga referencji z innych ramek (ala JPEG)



P – predictive – zdekodowanie makrobloku wymaga referencji z jednej ramki



B – bi-predictive – zdekodowanie makrobloku wymaga referencji z “dwóch” ramek



# Słowniczek - predykcja

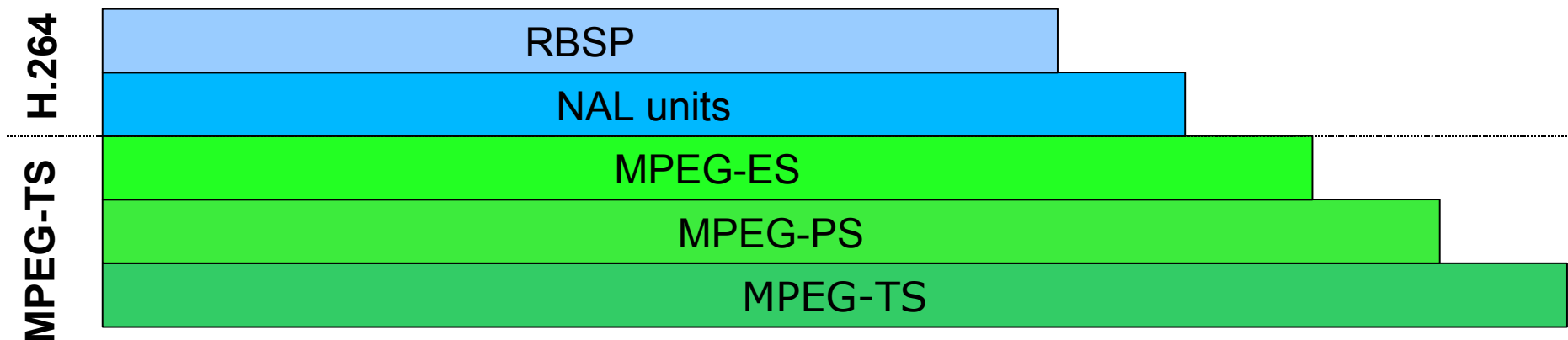
- **Predykcja** (czas.) - wyszukiwanie/odtworzenie podobieństw czasowych i/lub przestrzennych (zgodnie z ustaloną i dopuszczalną metodologią)
  - **Predykcja przestrzenna** (Intra)
  - **Predykcja czasowo-przestrzenna** (Inter) = estymacja/kompensacja ruchu
- **Predykcja** (rzecz.) - piksele (obraz) “uznane” za “podobne” do aktualnie kodowanego wycinka obrazu
- **Błąd predykcji** – różnica między obrazem oryginalnym i predykcją (może być ujemny)

Oryginał		Predykcja		Błąd predykcji			
2	2	2	2	0	0	-1	1
2	2	2	1	0	-2	1	0
2	1	1	1	1	0	1	1
1	1	1	1	-1	-2	0	0

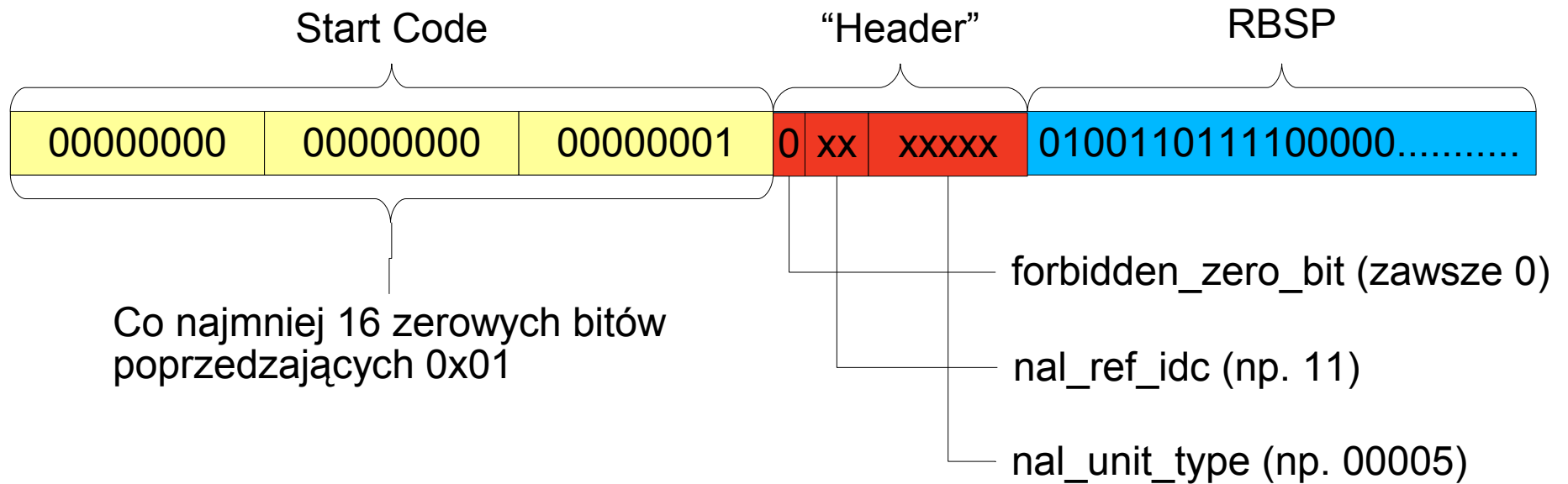
- Kodowaniu binarnemu podlega skwantowany i przekształcony błąd predykcji oraz metoda predykcji

# Enkapsulacja strumienia (przykład)

- **MPEG-TS** – transport DVB-S/T/C/H
- **MPEG-PS** – pojedynczy program
- **MPEG-ES** - “elementary stream”, dane dla dekodera (tutaj wideo)
- **NAL units** – Network Adaptation Layer – minimalnej długości nagłówek oraz escape'ing.
- **RBSP** – Raw Byte Sequence Payload – “czyste” dane do dekodowania strumienia H.264 – slice'y, meta-informacje, etc.



# Dekodujemy NAL



Co najmniej 16 zerowych bitów poprzedzających 0x01

nal_unit_type	Dane	Komentarz
1	slice	Dekodowanie obrazu
5	IDR	Dekodowanie obrazów IDR
6	SEI	Meta: timing, buffering period, pan-scan, etc.
7	SPS	
8	PPS	

# Dekodujemy slice

1. Kolejność działań (składnia zgodnie ze standardem):

```
seq_parameter_set_rbsp()  
pic_parameter_set_rbsp()  
slice_layer_without_partitioning_rbsp()  
  slice_header()  
  slice_data()  
  rbsp_slice_trailing_bits()
```

Bit string form	Range of codeNum
1	0
0 1 $x_0$	1-2
0 0 1 $x_1 x_0$	3-6
0 0 0 1 $x_2 x_1 x_0$	7-14
0 0 0 0 1 $x_3 x_2 x_1 x_0$	15-30
0 0 0 0 0 1 $x_4 x_3 x_2 x_1 x_0$	31-62
...	...

$$\text{codeNum} = 2^{\text{leadingZeroBits}} - 1 + \text{read\_bits}(\text{leadingZeroBits})$$

	C	Descriptor
slice_header() {		
first_mb_in_slice	2	ue(v)
slice_type	2	ue(v)
pic_parameter_set_id	2	ue(v)
frame_num	2	u(v)
if( !frame_mbs_only_flag ) {		
field_pic_flag	2	u(1)
if( field_pic_flag )		
bottom_field_flag	2	u(1)

# Dekodujemy slice (cd)

	C	Descriptor
slice_data() {		
if( entropy_coding_mode_flag )		
while( !byte_aligned() )		
<b>cabac_alignment_one_bit</b>	2	f(1)
CurrMbAddr = first_mb_in_slice * ( 1 + MbaffFrameFlag )		
moreDataFlag = 1		
prevMbSkipped = 0		
do {		
if( slice_type != I && slice_type != SI )		
if( !entropy_coding_mode_flag ) {		
<b>mb_skip_run</b>	2	ue(v)
prevMbSkipped = ( mb_skip_run > 0 )		
for( i=0; i<mb_skip_run; i++ )		
CurrMbAddr = NextMbAddress( CurrMbAddr )		
moreDataFlag = more_rbsp_data()		
} else {		
<b>mb_skip_flag</b>	2	ae(v)
moreDataFlag = !mb_skip_flag		
}		
if( moreDataFlag ) {		
if( MbaffFrameFlag && ( CurrMbAddr % 2 == 0		
( CurrMbAddr % 2 == 1 && prevMbSkipped ) )		
<b>mb_field_decoding_flag</b>	2	u(1)   ae(v)
macroblock_layer()	2   3   4	
}		

# Dekodujemy slice - CAVLC/Cabac

- W strumieniu możemy mieć symbole zakodowane różnymi metodami:
- **ae(v)** – symbol zakodowany enkoderem arytmetycznym (CABAC)
  - Współczynniki, flagi od poziomu **slice\_data**
- **ce(v)** – symbol zakodowany enkoderem zmiennej długości słowa (CAVLC)
  - Współczynniki
- **ue(v)** – symbol zakodowany enkoderem Exp-Golomb
  - Nagłówki oraz flagi i zmienne w przypadku braku CABAC'a
- **u(n)** – integer “n” bitowy (MSB)
  - Nagłówki, niektóre flagi w przypadku użycia CAVLC

# Dekodujemy slice - CAVLC

- Dla każdego bloku 4x4:

- Odczytujemy liczbę niezerowych współczynników z bloków sąsiadujących od góry i z lewej,
- na tej podstawie obliczamy parametr: **nC**

TrailingOnes (coeff token)	TotalCoeff (coeff token)	$0 \leq nC < 2$	$2 \leq nC < 4$	$4 \leq nC < 8$	$8 \leq nC$	$nC == -1$
0	0	1	11	1111	0000 11	01
0	1	0001 01	0010 11	0011 11	0000 00	0001 11
1	1	01	10	1110	0000 01	1
0	2	0000 0111	0001 11	0010 11	0001 00	0001 00
1	2	0001 00	0011 1	0111 1	0001 01	0001 10
2	2	001	011	1101	0001 10	001

- Używając powyższej tabeli określamy wartości **TrailingOnes** i **TotalCoeff** na podstawie
- bitów odczytanych ze strumienia
- Odczytujemy ze strumienia **level\_prefix** (kod zmiennej długości)
- Na podstawie powyższych wartości (i innych warunków) obliczamy zmienne
- **suffixLength** i **level\_suffix**
- Następnie obliczamy zmienną **levelCode** = (level\_prefix << suffixLength) + level\_suffix
- Ostatecznie dostajemy: **level** = (levelCode + 2) >> 1 lub **level** = (-levelCode - 1) >> 1

# Dekodujemy slice - CAVLC

- Obliczone współczynniki musimy połączyć z informacją na temat ich położenia (run-level)
- Odczytujemy zmienną **total\_zeros** ze strumienia
- Odczytujemy wartość **run\_before** dla każdego niezerowego współczynnika ze strumienia

total zeros	TotalCoeff( coeff token )						
	1	2	3	4	5	6	7
0	1	111	0101	0001 1	0101	0000 01	0000 01
1	011	110	111	111	0100	0000 1	0000 1
2	010	101	110	0101	0011	111	101
3	0011	100	101	0100	111	110	100
4	0010	011	0100	110	110	101	011

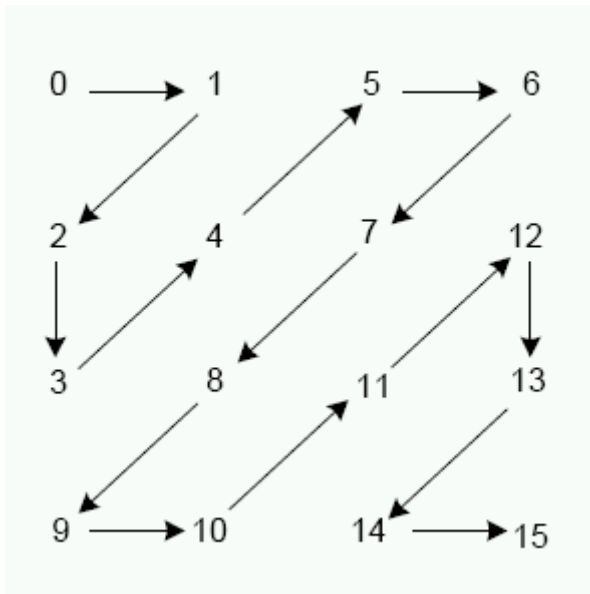
run_befor e	zerosLeft						
	1	2	3	4	5	6	>6
0	1	1	1 1	11	11	11	111
1	0	0 1	1 0	10	10	00 0	110
2	-	0 0	0 1	01	01 1	00 1	101
3	-	-	0 0	00 1	01 0	01 1	100

# Dekodujemy slice - CAVLC

- Na podstawie odczytanych zmiennych ustalamy pozycję każdego współczynnika l
- dostajemy wektor 15 (lub 16) współczynników

**coeffs** = [4, -2, 0, 0, -2, -1, 0, 0, -1, 1, 0, 0, 0, 1, -1, 0]

- Za pomocą odwrotnego zig-zag skanu umieszczamy otrzymany wektor w bloku 4x4



4	-2	-1	0
0	-2	0	0
0	-1	0	1
1	0	-1	0

# Dekodujemy slice - CABAC

## • Context-based Adaptive Binary Arithmetic Coding

- **9%-14%** lepsza wydajność (mniejszy strumień bitowy) niż w przypadku np. CAVLC dzięki:
  - Wybieraniu modelu prawdopodobieństwa dla każdego symbolu w zależności od kontekstu
  - Adaptacji estymat prawdopodobieństwa w zależności od statystyki lokalnej
  - Kodowaniu arytmetycznemu

## • Kodowanie:

- Kodowane są tylko decyzje binarne (0 lub 1)
- Symbole niebinarne (np. współczynniki, wektory ruchu) podlegają najpierw procesowi binaryzacji (zamiany na kod binarny) – proces podobny do zamiany na kod zmiennej długości
- **Dla każdego bin'a :**
  - wybór kontekstu – modelu prawdopodobieństwa dla **bin'ów** symbolu : model
  - przechowuje prawdopodobieństwo przyjmowania przez każdy bin wartości “1” lub “0”
  - Koder arytmetyczny koduje każdy bin w zależności od modelu prawdopodobieństwa
  - Aktualizacja modelu prawdopodobieństwa : np. jeśli zakodowany bin miał wartość “1” to zwiększana jest częstotliwość wystąpień “1”

# Dekodujemy slice – CABAC (cd)

## Przykład : MVDx (błąd predykcji wektora ruchu)

**MVDx**    **binaryzacja**

**0**        **0**  
**1**        **10**  
**2**        **110**  
**3**        **1110**  
**4**        **11110**  
**5**        **111110**  
**6**        **1111110**  
**7**        **11111110**  
**8**        **111111110**

Dla bin'u 1 możemy wybrać 3 modele w zależności od poprzednio zakodowanych wartości MVD dla bloków bezpośrednio po lewej (A) i ponad (B) aktualnego bloku

$$e_k = |MVDA| + |MVDB|$$

$$0 \leq e_k < 3 \quad \rightarrow \text{Model 0}$$

$$3 \leq e_k < 33 \quad \rightarrow \text{Model 1}$$

$$33 < e_k \quad \rightarrow \text{Model 2}$$

Bin	Model
1	0, 1, 2
2	3
3	4
4	5
5	6
$\geq 6$	6

- Po wybraniu modelu dla każdego bin'u enkodujemy bin'y : każdy model ma dwie estymaty
  - Prawdopodobieństwo, że bin jest "1"
  - Prawdopodobieństwo, że bin jest "0"
  - Estymaty określają zakresy, których następnie używa enkoder arytmetyczny
- Aktualizujemy model – jeśli dla bin'u 1 wybraliśmy model 2 i wartość binu wynosiła "0" to częstotliwość występowania "0" jest zwiększana, a co za tym idzie przy kolejnym wyborze tego modelu prawdopodobieństwo "0" będzie nieco wyższe.

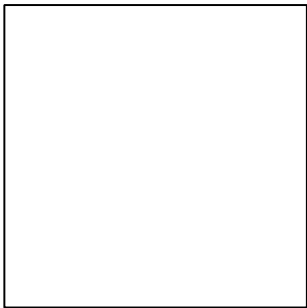


# Dekodujemy slice - predykcja (cd)

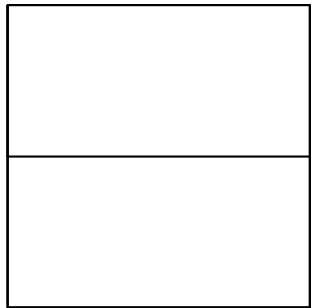
Makroblok Inter:

Partycje

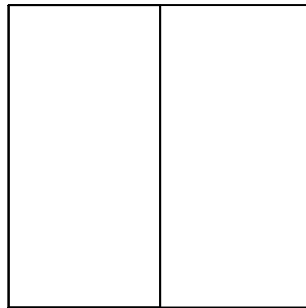
16x16



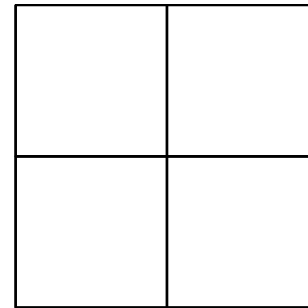
16x8



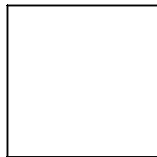
8x16



8x8



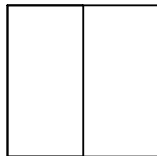
8x8



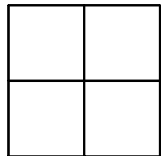
8x4



4x8

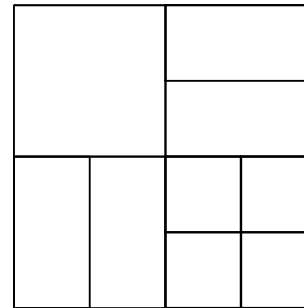


4x4



Sub-partycje dla trybu 8x8

np. →



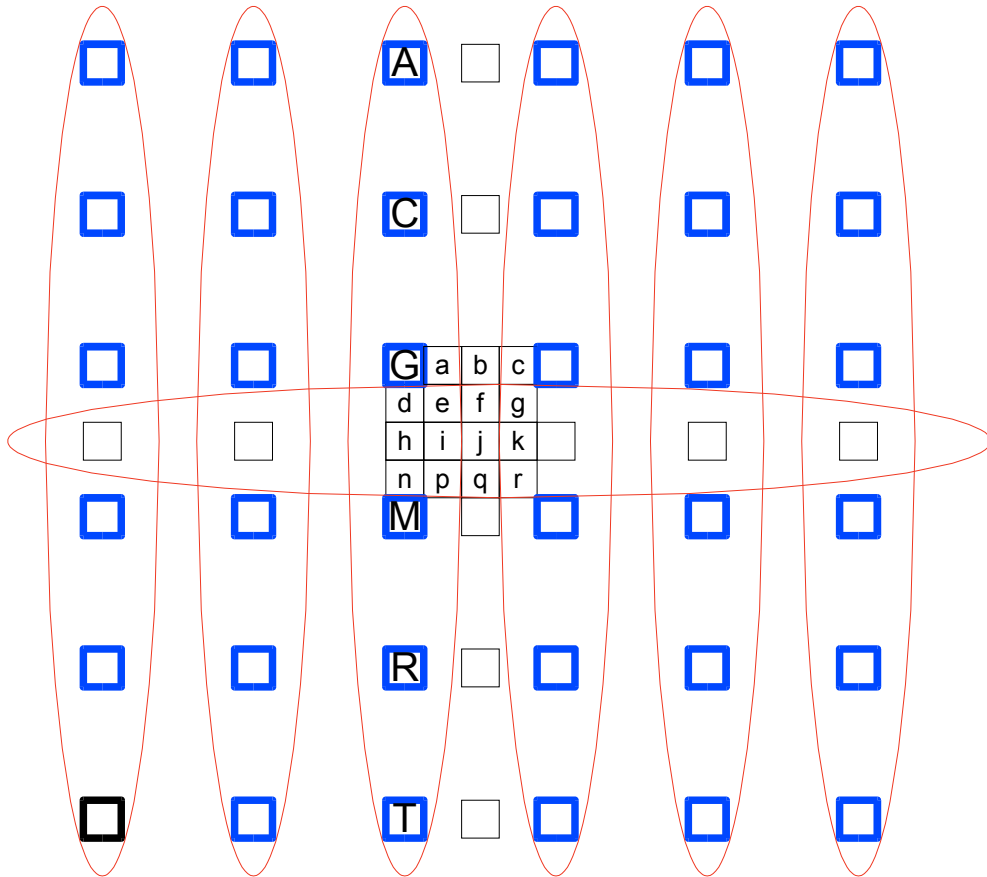
# Dekodujemy slice - predykcja (cd)

## **Makroblok Inter (interpolacja = predykcja Inter)**

Dla każdej partycji/sub-partycji:

- Odczytujemy indeks ramki referencyjnej ze strumienia
- Odczytujemy błąd predykcji dla wektorów ruchu ze strumienia
- Wykonujemy predykcję wektorów ruchu
- Otrzymujemy wektory ruchu (z dokładnością do  $\frac{1}{4}$  pixela)
- Liczymy predykcję dla danej partycji/subpartycji (pred):

# Dekodujemy slice - predykcja (cd)



6 tap filter: **1 -5 20 20 -5 1**

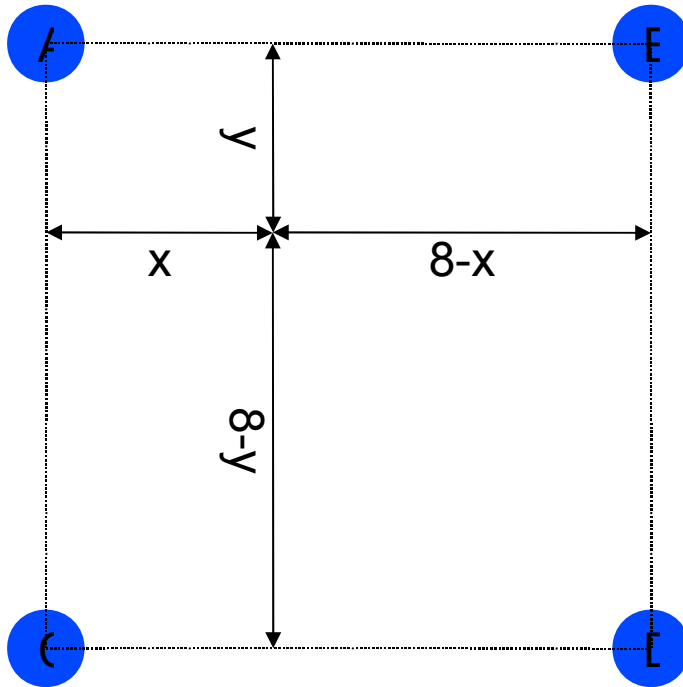
$$h = \begin{bmatrix} A & C & G & M & R & T \end{bmatrix} \times \begin{pmatrix} 1 \\ -5 \\ 20 \\ 20 \\ -5 \\ 1 \end{pmatrix}$$

Aby obliczyć piksel "i":

- Policzyć wszystkie "h"
- Policzyć "j"
- $i = (h + j + 1) \gg 1$

# Dekodujemy slice - predykcja (cd)

## Chrominancja inter (Cb/Cr)



$$\text{Pred} = ( A * (8 - x) * (8 - y) + \\ B * (x) * (8 - y) + \\ C * (8 - x) * y + \\ D * x * y + 32 ) \gg 6$$

# Transformacja odwrotna i skalowanie

## • Transformacja odwrotna dla DC Intra 16x16

- Dla każdego makrobloku odczytujemy 16 współczynników DC (c00-c33, 16bit każdy)
- ze strumienia i liczymy przekształcenie odwrotne

$$f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}.$$

- Liczymy odwrotną kwantyzację

$$\text{LevelScale}(m, i, j) = \begin{cases} v_{m0} & \text{for } (i, j) \in \{(0,0), (0,2), (2,0), (2,2)\}, \\ v_{m1} & \text{for } (i, j) \in \{(1,1), (1,3), (3,1), (3,3)\}, \\ v_{m2} & \text{otherwise;} \end{cases}$$

$$v = \begin{bmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{bmatrix}.$$

$$\text{dc}Y_{ij} = (f_{ij} * \text{LevelScale}(QP_Y \% 6, 0, 0)) \ll (QP_Y / 6 - 2), \quad \text{Dla } QP_Y \geq 12$$

$$\text{dc}Y_{ij} = (f_{ij} * \text{LevelScale}(QP_Y \% 6, 0, 0) + 2^{1-QP_Y/6}) \gg (2 - QP_Y / 6), \quad \text{Dla } QP_Y < 12$$

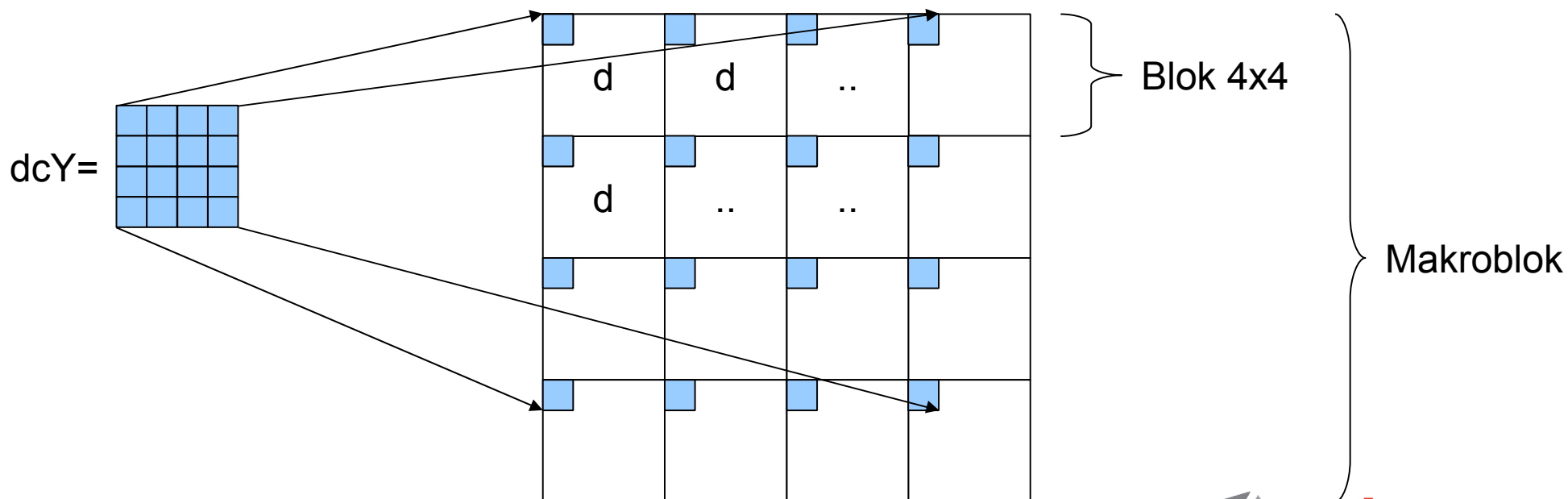
# Transformacja odwrotna i skalowanie

## Transformacja odwrotna dla tzw. “residual data” (błąd predykcji)

- Dla każdego zakodowanego bloku 4x4 danego makrobloku odczytujemy pozostałe
- współczynniki ze strumienia ( $C_{ij}$ ) (16 dla Intra4x4 i Inter oraz 15 dla Intra16x16)
- Skalujemy odczytane współczynniki

$$d_{ij} = (c_{ij} * \text{LevelScale}(qP \% 6, i, j)) \ll (qP / 6),$$

- Dla Intra16x16 wstawiamy poprzednio obliczone współczynniki DC na pozycję (0,0) do macierzy  $d$



# Transformacja odwrotna i skalowanie

## Transformacja odwrotna dla tzw. “residual data” (błąd predykcji)

- Transformacja odwrotna jest równoważna następującemu wyrażeniu:

$$\mathbf{h} = \mathbf{A} \mathbf{d} \mathbf{A}^T$$

$\mathbf{d}$  = blok 4x4

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{pmatrix}$$

- Finalny wyniki transformaty odwrotnej zapisujemy w postaci:

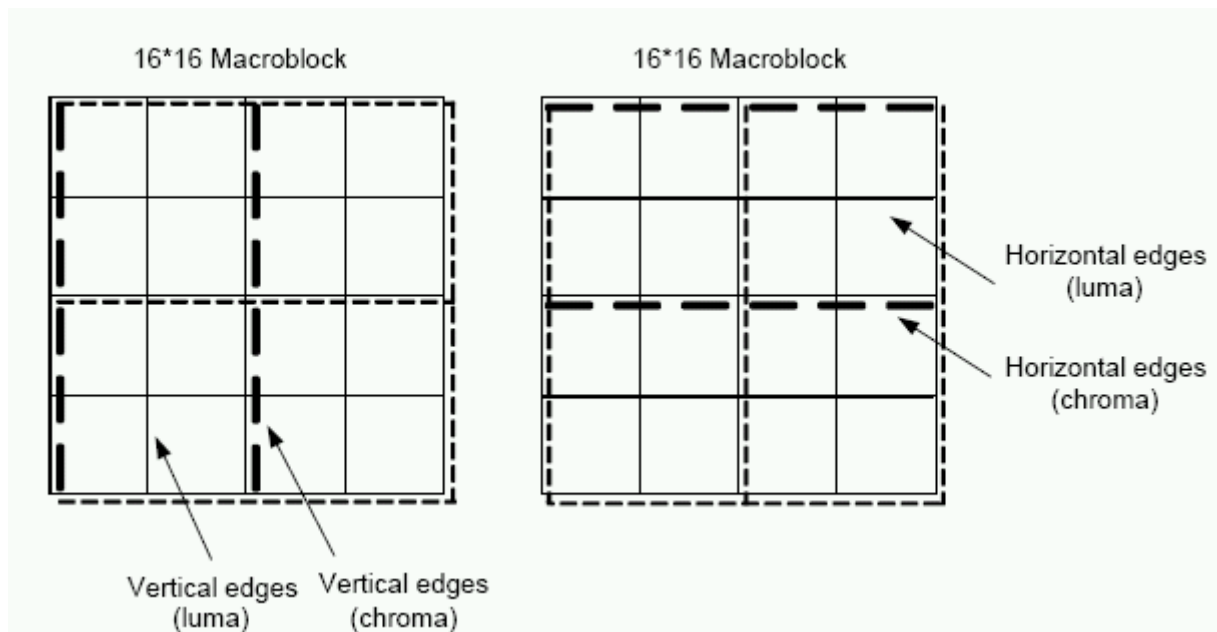
$$r_{ij} = (h_{ij} + 2^5) \gg 6 \quad \text{with } i, j = 0..3$$

- Dodajemy predykcje do błędu predykcji (residual) – rekonstruujemy obraz

$$u_{ij} = \text{Clip1}(\text{pred}_L[xO + j, yO + i] + r_{ij}) \quad i, j = 0..3$$

# Deblocking filter

- Operację filtrowania obrazu wykonujemy gdy wszystkie makrobloki zostały zrekonstruowane
- Filtrowana jest każda krawędź każdego bloku 4x4 za wyjątkiem krawędzi obrazu oraz krawędzi, dla których filtracja jest wyłączona (flaga **disable\_deblocking\_filter\_idc** w **slice\_header**)
- Filtrujemy zarówno luminancję jak i chrominancję
- Dla każdego makrobloku najpierw filtrujemy wszystkie krawędzie pionowe, a następnie poziome
- Do filtracji używamy pikseli, które uprzednio mogły już być przefiltrowane w sąsiednich makroblokach lub obecnym makrobloku (przez filtrację pionową)

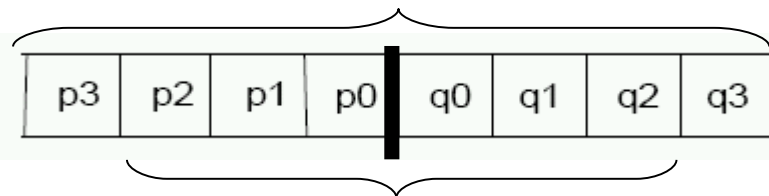


# Deblocking filter (cd)

• Proces filtracji zależny jest w głównej mierze od następujących zmiennych:

- field\_pic\_flag
  - MbaffFrameFlag oraz czy aktualny makroblok jest “z przeplotem”
  - disable\_deblocking\_filter\_idc (filtracja tylko w obrębie slice'u, normalna bądź wyłączona)
  - Typ aktualnego makrobloku (Intra, Inter, partycje, subpartycje) i sąsiadujących makrobloków
  - Czy blok, dla którego obliczamy krawędź posiada niezerowe współczynniki (transformaty)
  - Indeksy ramek referencyjnych dla aktualnego i sąsiadujących makrobloków
  - Różnica w długości wektorów ruchu dla aktualnego i sąsiadujących makrobloków (P, B)
  - QP dla aktualnego i sąsiadujących makrobloków
- Między innymi na podstawie powyższych zależności obliczamy siłę filtru dla danej krawędzi: **bS**
- Filtr modyfikujący piksele zależny jest od parametru **bS**

Piksele używane do filtracji



Piksele (potencjalnie) zmienione podczas filtracji

# Deblocking filter (cd)

- Na podstawie  $Q_p$  dla aktualnego i sąsiadujących makrobloków ustalamy
- parametry filtru  $\alpha$  i  $\beta$  (alfa i beta) (odczytujemy z tablicy)
- Na podstawie  $\alpha$  i  $bS$  odczytujemy parametr  $t_0$  (z tablicy)
- Przykład filtracji dla  $bS < 4$ :

$$a_p = \text{Abs}(p_2 - p_0)$$

$$a_q = \text{Abs}(q_2 - q_0)$$

$$\text{ChromaEdgeFlag} == 0 ? t_C = t_{C0} + ((a_p < \beta) ? 1 : 0) + ((a_q < \alpha) ? 1 : 0) : t_C = t_{C0} + 1$$

$$\Delta = \text{Clip3}(-t_C, t_C, (((q_0 - p_0) \ll 2) + (p_1 - q_1) + 4) \gg 3)$$

$$p'_0 = \text{Clip1}(p_0 + \Delta)$$

$$q'_0 = \text{Clip1}(q_0 - \Delta)$$

$$\text{chromaEdgeFlag} == 0 \ \&\& \ a_p < \beta ? p'_1 = p_1 + \text{Clip3}(-t_{C0}, t_{C0}, (p_2 + ((p_0 + q_0 + 1) \gg 1) - (p_1 \ll 1)) \gg 1) : p'_1 = p_1$$

$$\text{chromaEdgeFlag} == 0 \ \&\& \ a_q < \beta ? q'_1 = q_1 + \text{Clip3}(-t_{C0}, t_{C0}, (q_2 + ((p_0 + q_0 + 1) \gg 1) - (q_1 \ll 1)) \gg 1) : q'_1 = q_1$$

$$p'_2 = p_2$$

$$q'_2 = q_2$$

# Problemy implementacyjne

- Ekstremalnie trudna (i żmudna) implementacja
  - Skomplikowany data-flow – mnogość warunków (250 stron A4 opisów i wzorów dotyczących dekodowania)
  - Nietrywialny error-resilience oraz error-concealment
  - Bugi wychodzą nawet po latach (np. dla “złośliwych” danych)
- Problemy sprzętowe
  - Dla “optymalnych” implementacji 4x więcej CPU niż MPEG-2, dla kiepskich jeszcze więcej ;)
  - (todo: life przykład użycia pamięci dla D1)
  - Słaba wydajność I-Cache (dużo kodu, dużo skoków warunkowych)
  - Słaba wydajność D-Cache (L1, L2, L3..) - predykcja Inter, “2D” transfery
  - Na dedykowanych procesorach (np. DSP) konieczność bezpośredniego użycia engine'u DMA = projektowanie software'owych potoków, problemy z synchronizacją
  - Użycie koprocessorów = ekstremalnie długie potoki software'owe (skomplikowane warunki brzegowe, długi epilog/prolog)

# Q & A

Proszę o pytania :)